A PIPELINED ARCHITECTURE FOR GLOBAL ANALYSIS AND INDEX BUILDING

BACKGROUND OF THE INVENTION

1. Field of the Invention

[0001] The present invention is related to a pipelined architecture for Global analysis and index building.

2. Description of the Related Art

[0002] The World Wide Web (also known as WWW or the "Web") is a collection of some Internet servers that support Web pages that may include links to other Web pages. A Uniform Resource Locator (URL) indicates a location of a Web page. Also, each Web page may contain, for example, text, graphics, audio, and/or video content. For example, a first Web page may contain a link to a second Web page. When the link is selected in the first Web page, the second Web page is typically displayed.

[0003] A Web browser is a software application that is used to locate and display Web pages. Currently, there are billions of Web pages on the Web.

[0004] Web search engines are used to retrieve Web pages on the Web based on some criteria (e.g., entered via the Web browser). That is, Web search engines are designed to return relevant Web pages given a keyword query. For example, the query "HR" issued against a company intranet search engine is expected to return relevant pages in the intranet that are related to Human Resources (HR). The Web search engine uses indexing techniques that relate search terms (e.g., keywords) to Web pages.

[0005] Global analysis computations may be described as extracting properties from a global view of documents in a corpus (e.g., documents available on the Web). One example of a global analysis computation is the page rank computation. A page rank computation takes as input a directed graph in which every document in the corpus is a node and every hyperlink between documents is an edge. Then, the page rank computation produces as output a global rank for each document in the corpus. Other examples of global analysis computations are duplicate detection (i.e., the identification

of pages with similar or the same content) and template detection (i.e., identification of which parts of a Web page are part of a site template).

[0006] Search engines that use global analysis computations typically need to have the output of these computations ready before indexing the corpus. For instance, rank values computed by page rank may be used to determine the order of documents in the index, and the results of the duplicate detection computation may be used to filter out which documents should not be indexed. Having to perform all global analysis computations before the creation of the search indices is a problem in scenarios where freshness requirements impose constraints on the time allowed for index creation. In general, global computations are costly, since their computational time is proportional to the number of documents in the corpus, which in the case of the Web or some textual and biological databases is very large.

[0007] Additionally, conventional index structures designed for large scale search engines are not well tuned for incremental updates. Thus, incrementally updating an index is expensive in conventional systems.

[0008] Thus, there is a need for improved global analysis and index building.

SUMMARY OF THE INVENTION

[0009] Provided are a method, system, and program for building an index in which global analysis computations and index creation are pipelined, wherein the global analysis computations share intermediate results.

BRIEF DESCRIPTION OF THE DRAWINGS

Referring now to the drawings in which like reference numbers represent corresponding parts throughout:

FIG. 1 illustrates, in a block diagram, a computing environment in accordance with certain implementations of the invention.

FIG. 2 illustrates a data store and data structures in accordance with certain implementations of the invention.

FIG. 3 illustrates an index build process implemented by an indexing component in accordance with certain implementations of the invention.

FIG. 4 illustrates a delta index build process implemented by a delta indexing component in accordance with certain implementations of the invention.

FIG. 5 illustrates a process for merging a store and a delta store in accordance with certain implementations of the invention.

FIG. 6A illustrates logic for an index build process in accordance with certain implementations of the invention.

FIG. 6B illustrates, in a block diagram, a high-level flow of an index build process in accordance with certain implementations of the invention.

FIG. 6C illustrates auxiliary data structures that may be used by global analysis computations in accordance with certain implementations of the invention.

FIG. 7A illustrates logic for an index build process in accordance with certain alternative implementations of the invention.

FIG. 7B illustrates, in a block diagram, a high-level flow of an index build process in accordance with certain alternative implementations of the invention.

FIG. 8 illustrates processing using current global analysis results versus using slightly older global analysis results in accordance with certain implementations of the invention.

FIG. 9A illustrates logic for a global analysis process in accordance with certain implementations of the invention.

FIG. 9B illustrates, in a block diagram, a flow of a global analysis process in accordance with certain implementations of the invention.

FIG. 10 illustrates an architecture of a computer system that may be used in accordance with certain implementations of the invention.

DETAILED DESCRIPTION

[0010] In the following description, reference is made to the accompanying drawings which form a part hereof and which illustrate several implementations of the present invention. It is understood that other implementations may be utilized and structural and

operational changes may be made without departing from the scope of the present invention.

[0011] Implementations of the invention provide an architecture in which global analysis computations are pipelined so that intermediate results may be shared between different sets of global analysis computations and where lagging global analysis information is used for creating search indices. The term "lagging" may be described as performing processing with results of recent processing of documents existing at a certain point in time, but not necessarily processing that has taken into account documents existing at a current point in time.

[0012] FIG. 1 illustrates, in a block diagram, a computing environment in accordance with certain implementations of the invention. A client computer 100 is connected via a network 190 to a server computer 120. The client computer 100 may comprise any computing device known in the art, such as a server, mainframe, workstation, personal computer, hand held computer, laptop telephony device, network appliance, etc. The network 190 may comprise any type of network, such as, for example, a Storage Area Network (SAN), a Local Area Network (LAN), Wide Area Network (WAN), the Internet, an Intranet, etc. The client computer 100 includes system memory 104, which may be implemented in volatile and/or non-volatile devices. One or more client applications 110 and a viewer application 112 may execute in the system memory 104. The viewer application 112 provides an interface that enables searching of a set of documents (e.g., stored in one or more data stores 170. In certain implementations, the viewer application 112 is a Web browser.

[0013] The server computer 120 includes system memory 122, which may be implemented in volatile and/or non-volatile devices. A information retrieval system 130 executes in the system memory 122. In certain implementations, the search engine includes a crawler component 132, a global analysis component 134, an indexing component 142, and a delta indexing component 144. The global analysis component 134 includes a static rank component 136, a duplicate detection component 138, an anchor text component 140. Although components 132, 136, 134, 138, 140, 142, and 144 are illustrated as separate components, the functionality of components 132, 136, 134,

138, 140, 142, and 144 may be implemented in fewer or more or different components than illustrated. Additionally, the functionality of the components 132, 136, 134, 138, 140, 142, and 144 may be implemented at a Web application server computer or other server computer that is connected to the server computer 120. Additionally, one or more server applications 160 execute in system memory 122.

[0014] The server computer 120 provides the client computer 100 with access to data in at least one data store 170 (e.g., a database). Although a single data store 170 is illustrated, for ease of understanding, data in data store 170 may be stored in data stores at other computers connected to server computer 120.

[0015] Also, an operator console 180 executes one or more applications 182 and is used to access the server computer 120 and the data store 170.

[0016] The data store 170 may comprise an array of storage devices, such as Direct Access Storage Devices (DASDs), Just a Bunch of Disks (JBOD), Redundant Array of Independent Disks (RAID), virtualization device, etc. The data store 170 includes data that is used with certain implementations of the invention.

[0017] In certain implementations of the invention, global analysis computations are performed by the global analysis component 134 in a pipelined manner with the use of lagging information and delta indices for an index build process performed by the indexing component 142. The global analysis computations described herein provide examples of possible global analysis computations, but such examples are not intended to limit the scope of the invention. The techniques of the invention are applicable to any global analysis computations.

[0018] FIG. 2 illustrates a data store and data structures in accordance with certain implementations of the invention. Data store 200 may be one of data stores 170 (FIG. 1). Data store 200 includes a store 210, an index 220, a delta store 230, and a delta index 240. The store 210, index 224, delta store 230, and delta index 240 may be used in an information retrieval system 130. The store 210, index 224, delta store 230, and delta index 240 are data structures used for index creation and global analysis processes in accordance with certain implementations of the invention.

[0019] The store 210 is a repository for a tokenized version of each document in a corpus of documents to be used for the information retrieval system 130. For example, the documents may be Web pages obtained at a certain point in time or the documents may be files in a directory obtained at a certain point in time. The documents are parsed and tokenized before being put in the store 210. Parsing and tokenizing may be described as identifying words (tokens) from text in the documents that are to be indexed. Parsed documents are typically small, so they may be aggregated into bundles in the store 210 to improve Input/Output (I/O) performance. Each bundle corresponds to a single file and multiple documents. A document in the store 210 may be found using a locator. The locator may be described as a reference to a bundle followed by a reference within the bundle. Also, implementations of the invention provide an Application Programming Interface (API) that may be used to retrieve a document from the store 210, given the locator of the document.

[0020] Additionally, attribute-value pairs may be used to store data and metadata about a document. The data in the store 210 is used by the global analysis component 134 and by the indexing component 142. Attributes include, for example, a TEXT attribute for the tokenized text of a document, and attributes that are used as input for global analysis computations, such as an ANCHORTEXT attribute for the source anchor text of a document. An anchor may be described as a path or link to a document (e.g., a URL), while anchor text may be described as text associated with the path or link (e.g., a URL) that points to a document. In certain implementations, anchor text is text that labels or encloses hypertext text links in Web documents. Implementations of the invention provide an API to retrieve the value of a given attribute for a document from the store 210.

[0021] In certain implementations, the bundles are organized for efficient storage and retrieval for particular attributes. The store 210 may be implemented in various ways. That is, the implementation of the store 210 is independent from the pipelined architecture described for implementations of the invention, as long the functionality of retrieving attributes from documents is supported.

[0022] The indexing component 142 builds an index 220 from the store 210. The index 220 is used to determine which documents are chosen as "hits" when a query for a set of documents is received with certain criteria. Different information retrieval system 130 may use different index structures, such as an inverted text index over the documents content. Implementations of the invention are applicable to any type of index structure.

[0023] The information retrieval system 130 periodically updates the store 210 and index 220 with new documents. In certain implementations, the term "new" documents is used to refer to newly generated documents and to updated existing documents. This is especially useful in environments in which documents are constantly being created or updated. Additionally, certain implementations of the invention use a delta store 230 to accumulate changes to the store 210 and use a delta index 240 as an index over the delta store 230. This allows the index 220 to be rebuilt using an efficient batch process. The delta store 230 and delta index 240 mirror the structure and functionality of the store 210 and index 220, respectively. To make newly documents visible, queries requesting documents are run over both the index 220 and the delta index 240. The delta store 230 is periodically merged with the store 210, after which the index 220 is rebuilt, and the delta store 230 and the delta index 240 are reset. Resetting may be described as clearing the data structures of current content.

[0024] FIG. 3 illustrates an index build process 300 implemented by the indexing component 142 in accordance with certain implementations of the invention. The index build process 300 takes as input the current version of store 210 (labeled $Store_i$ in FIG. 3) and the current version of delta store 230 (labeled DeltaStore in FIG. 3). The index build process 300 then merges the current version of store 210 with the current version of delta store 230 and generates a new version of store 210 (labeled $Store_{i+1}$ in FIG. 3) and a new index 220 (labeled $Index_{i+1}$ in FIG. 3). The store 210 and index 220 are generated together in time in this manner. As $Store_{i+1}$ is generated from the current store 210 and the delta store 230, garbage collection takes place. Garbage collection may be described as a policy for ensuring that newer versions of documents replace older versions of documents in the store 210. For example, if document D in $Store_i$ has been replaced by a newer version of document D in the delta store 230 (called document D'), then only

document D' appears in $Store_{i+1}$. After the index build process 300 has finished, $Store_{i+1}$ and $Index_{i+1}$ are copied, for example, to an appropriate data store 170, and the delta store 230 and delta index 240 are reset. A single scan of $Store_i$ and the DeltaStore is sufficient to perform garbage collection and generate $Store_{i+1}$ and $Index_{i+1}$.

[0025] The delta store 230 and the delta index 240 are also generated together in time, but at a faster rate than the store 210 and the index 220. Thus, DeltaStore has no subscript in FIG. 3. FIG. 4 illustrates a delta index build process 400 implemented by the delta indexing component 144 in accordance with certain implementations of the invention. The delta index build process 400 takes as input a version of the delta store (labeled $DeltaStore_j$ in FIG. 4) and new documents. The delta index build process 400 merges $DeltaStore_j$ with the new documents to generate a new version of delta store (labeled $DeltaStore_{j+1}$ in FIG. 4) and a new version of delta index (labeled $DeltaIndex_{j+1}$ in FIG. 4). New documents may be stored in the same manner as documents in the delta store 230. Consequently, the process shown in FIG. 4 is similar to the one shown in FIG. 3, with new documents effectively acting as a delta to the delta store 230. A single scan of the new documents and $DeltaStore_j$ is sufficient to do garbage collection and generate $DeltaStore_{j+1}$ and $DeltaIndex_{j+1}$.

[0026] FIG. 5 illustrates a process for merging a store 210 and delta store 230 in accordance with certain implementations of the invention. Store 210 is labeled $Store_i$ 510 and the delta store 230 is labeled DeltaStore 520 in FIG. 5. In certain implementations, for the processing of FIG. 5, data associated with a document (i.e., attribute value pairs) are grouped together into a single binary object. In the example of FIG. 5, documents D1 and D5 in $Store_i$ 510 have been replaced by more recent versions documents D1' and D5', respectively, in DeltaStore 520. Thus, documents D1 and D5 are garbage collected as a newer version of the data store (labeled $Store_{i+1}$ 530 in FIG. 5) is generated. In certain implementations, copying a bundle from the DeltaStore 520 to $Store_{i+1}$ 530 may be done with a file rename, depending on whether any meta-data in the bundle is to be updated.

[0027] The process illustrated in FIG. 5 may also be used to merge the delta store 230 and new documents. However, multiple versions of the same document may appear in

the new documents. To ensure that the most recent version of a document is merged into the delta store 230, new documents are scanned in Last In First Out (LIFO) order.

[0028] In certain implementations, in FIG. 5, as a bundle in DeltaStore 520 is scanned, the bundle is copied to $Store_{i+1}$ 530, with a filter being set to indicate that the documents in this bundle have been "seen". After DeltaStore 520 is scanned, $Store_i$ 510 is scanned. As a bundle is scanned in $Store_i$ 510, the filter is probed to determine whether any documents in that bundle have been "seen" (i.e., copied from the DeltaStore 520 to $Store_{i+1}$ 530). Any "seen" documents may be skipped in $Store_i$ 510 (i.e., not copied to $Store_{i+1}$). In certain implementations, a hash table may be used in conjunction with the filter. Structure 540 represents a filter that optionally has an associated hash table.

[0029] To avoid multiple scans of the current store 210 during index build, $Store_{i+1}$ and $Index_{i+1}$ are generated in parallel. This is accomplished by feeding the bundles of $Store_{i+1}$ into the sort for $Index_{i+1}$ before they are copied to storage (e.g., disk). The final phase of building $Index_{i+1}$, which consists of merging sorted runs, takes place after $Store_{i+1}$ is copied to storage (e.g., disk). The delta index 240 is built in a similar way, that is, using a single scan of the delta store 230 and new documents. Certain implementations of the invention provide a store merge API that marks two stores as merged and does not do any garbage collection until the index build process initiates a scan of the merged store.

[0030] FIG. 6A illustrates logic for an index build process in accordance with certain implementations of the invention. FIG. 6B illustrates, in a block diagram, a high-level flow of an index build process in accordance with certain implementations of the invention. In FIG. 6A, control begins at block 600 and documents that are to be indexed by the search engine 130 are obtained. In certain implementations, the documents are published or pushed (e.g., as may be the case with newspaper articles) to the indexing component 142. In certain implementations, the crawler component 132 discovers, fetches, and stores the documents. At a subsequent time, in block 602, the crawler component 132, discovers, fetches, ad stores new documents in delta store 230.

[0031] In block 604, the global analysis component 134 performs global analysis. For example, the global analysis component 134 may perform multiple global analysis computations, such as duplicate detection, anchor text analysis, and static ranking. In

certain implementations, the global analysis component 134 performs the duplicate detection, anchor text analysis, and static ranking directly, and, in certain implementations, the global analysis component 134 invokes the duplicate detection 138, anchor text 140, and static rank 136 components to perform this processing.

[0032] FIG. 6C illustrates auxiliary data structures that may be used by global analysis computations in accordance with certain implementations of the invention. The auxiliary data structures include a duplicates table 670, an anchor text table 673, and a rank table 674. The duplicates table 670 identifies documents that are duplicates of each other and identifies a "master" among duplicates. The anchor text table 672 collects anchor text pointing to each document in the store 210. The rank table associates a static rank with each document in the store 210. Thus, the global analysis component 134 is capable of performing any global analysis computations used by the information retrieval system 130, using data structures used by each of these global analysis computations. The global analysis computations and data structures presented herein are just examples of possible scenarios. The global analysis computations and data structures presented herein are useful to illustrate how the global analysis computations are integrated in the index build process and how intermediate results may be shared among global analysis computations in a pipelined manner.

[0033] The duplicates table 670, anchor text table 673, and rank table 674 are derived from a global analysis of the documents in the store 210. Although the duplicates table 670, anchor text table 673, and rank table 674 are illustrated as single entities, one or more of the duplicates table 670, anchor text table 673, and rank table 674 may be implemented as one or more tables in storage (e.g., disk). A subscript used with each data structure denotes which version of the store 210 that the data structure reflects. For example, $Rank_{i+1}$ corresponds to the ranking of documents in $Store_{i+1}$.

[0034] After the global analysis component performs processing, the indexing component builds an index 220 (block 606) and builds a delta index 240 (block) 608.

[0035] With reference to FIG. 6B, a global analysis process 650 implemented by the global analysis component 134 takes $Store_i$ and $DeltaStore_i$ as input and outputs $Rank_{i+1}$, $AnchorText_{i+1}$, and Dup_{i+1}. The index build process 652 implemented by the indexing

component 142 then takes this information, along with $Store_i$ and $DeltaStore_i$, and generates $Store_{i+1}$ and $Index_{i+1}$. During this process, garbage collection is performed on $Store_i$. After $Store_{i+1}$ and $Index_{i+1}$ have been generated, the delta index build process 654 implemented by the delta indexing component 144 may be resumed. The delta index build process cycles at its own rate, generating a new delta store 230 and delta index 240 each cycle.

[0036] The performance of the index build process is largely bound by the time to do a scan of the store 210 and to perform the global analysis computations, which might be quite expensive. Two scans of the store 210 are required in FIG. 6B. One scan is to perform global analysis computations and another scan is to build the index. Certain implementations of the invention reduce the time for the index build process by modifying the flow for index build. In particular, certain implementations of the invention use $Rank_i$, $AnchorText_i$, and Dup_i to build $Index_{i+1}$ rather than $Rank_{i+1}$, $AnchorText_{i+1}$, and Dup_{i+1}. In other words, a slightly older ("stale") global analysis (GA_i) is used to build $Index_{i+1}$ rather than GA_{i+1}.

[0037] FIG. 7A illustrates logic for an index build process in accordance with certain alternative implementations of the invention. FIG. 7B illustrates, in a block diagram, a high-level flow of an index build process in accordance with certain alternative implementations of the invention. In FIG. 7A, control begins at block 700 with the crawler component 132 discovering, fetching, and storing documents in store 210. At a subsequent time, in block 702, the crawler component 132, discovers, fetches, ad stores new documents in delta store 230.

[0038] In block 704, the global analysis component 134 performs global analysis. In block 706, the indexing component 142 builds an initial index using slightly older input data. In block 708, the global analysis component 134 performs global analysis, while the delta index component 144 builds a new delta store and a new delta index.

[0039] With reference to FIG. 7B, the index build process 750 implemented by the indexing component 142 uses GA_i to build $Index_{i+1}$ rather than using GA_{i+1} to build $Index_{i+1}$. In certain implementations, there may be some loss in index precision, but the

loss is likely to be small since information, such as document rank, is unlikely to change drastically from GA_i to GA_{i+1}.

[0040] In FIG. 7B, the index build process 750 produces "raw" versions of $AnchorText_{i+1}$, and Dup_{i+1}. This information is extracted from $Store_i$ and DeltaStore as they are scanned. The term "raw" indicates that more processing is needed to produce the final version. For example, because of duplicates, the raw $AnchorText_{i+1}$ may contain multiple copies of the same anchor text. The global analysis process 752 implemented by the global analysis component 134 refines the raw information and produces GA_{i+1}.

[0041] In FIG. 7B, the anchor text for $Store_{i+1}$ is aggregated into $AnchorText_{i+1}$. The anchor text remains there, and the store is not later updated to add each document's anchor text to its bundle. This is avoided as it may require updating many bundles in the store, which is expensive in terms of resources that are used.

[0042] In addition to cutting the number of store 210 scans in half, using slightly older global analysis results (GA) also allows the delta index build process 754 to be done in parallel with the global analysis process 752. This allows the cycle time of the index build process to be reduced.

[0043] FIG. 8 illustrates processing using current global analysis results 800 versus using slightly older global analysis results 820 in accordance with certain implementations of the invention. As illustrated in FIG. 8, if the current global analysis results are used, then the delta index build process is stopped during the global analysis process, otherwise the global analysis would not reflect the current contents of the delta store. In contrast, by using a slightly older ("stale") global analysis results, the delta index build process may be performed in parallel with the global analysis process.

[0044] In order to generate the first index ($Index_0$), implementations of the invention run the index build process in FIG. 7B twice, with the appropriate inputs and outputs set to null each time. More specifically, building the first index begins with the first store ($Store_0$). In the first iteration, $Store_0$ is input to the indexing component 142, which outputs raw Dup_0 and raw $AnchorText_0$. These are input to the global analysis component 134, which outputs Dup_0, $AnchorText_0$, and $Rank_0$. Then, on the second iteration, $Store_0$,

$Rank_0$, Dup_0, and $AnchorText_0$ are input to the indexing component 142, which generates $Index_0$.

[0045] FIG. 9A illustrates logic for a global analysis process in accordance with certain implementations of the invention. FIG. 9B illustrates, in a block diagram, a flow of the global analysis process in accordance with certain implementations of the invention. In FIG. 9A, control begins at block 900 with duplicate detection 950 being performed using as input raw Dup_{i+1} and GA_i and outputting Dup_{i+1}. In block 902, anchor text analysis 952 is performed using as input $AnchorText_{i+1}$ and GA_i and outputs $AnchorText_{i+1}$. In block 904, static ranking 954 is performed with input $Rank_{i+1}$ and GA_i and outputting $Rank_{i+1}$.

[0046] With reference to FIG. 9B, the global analysis process inputs raw versions of $AnchorText_{i+1}$, and Dup_{i+1} and outputs $Rank_{i+1}$ along with $AnchorText_{i+1}$, and Dup_{i+1}. The duplicate detection produces Dup_{i+1} by sorting the raw version of Dup_{i+1} on, for example, a destination URL. Dup_{i+1} identifies a single "master" URL for each set of duplicate documents. This information is used by the anchor text analysis 952, which replaces destination and source URLs by their master URLs, so that duplicate anchor text may be kept out of the index. A projection on $AnchorText_{i+1}$ is then used to produce raw $Rank_{i+1}$ (e.g., in the form of a link graph ($Link_{i+1}$)) of destination and source URLs. The static rank component 136 computes $Rank_{i+1}$ from $Link_{i+1}$. Thus, common intermediate results are shared between global analysis computations.

[0047] Thus, when creating an index, global analysis need not yet be finished. Instead, the index build process uses the results of a recent global analysis process that has already finished. In certain implementations, for documents that have changed since the global analysis was computed, the change may be ignored and the available results may be used, while for new documents for which no results are available, a default value (e.g. a minimum or average value) may be used. That is, although document content may have changed, the already computed information, e.g. rank, may be used without loss of precision.

[0048] Lagging global analysis may be used in different scenarios. For example, if global analysis is slower than an index build, but fast enough to run between two index

builds, index build for iteration N may use the global analysis results from iteration (N-1). Hence global analysis for iteration N may run concurrently. Moreover, if global analysis takes longer than the time between two index builds, global analysis may run continuously in the background. Whenever an iteration of global analysis finishes, another iteration starts over with the most recent data. In certain implementations, global analysis may be skipped for a few iterations of the index build. The indexing component 142 then uses the most recent complete global analysis results.

[0049] A high frequency of index builds are achieved to fulfill strict freshness requirements, at the temporary cost of some precision in the global analysis. Also, the architecture allows new documents to be indexed in smaller indices (delta indices) that are queried together with the main indices. In certain implementations, global analysis is not performed for the delta indices in the first version of those indices. Thus, when the freshness requirements are strict compared to the time required by the global analysis computations, new documents may be indexed and retrieved before their global information is computed.

[0050] In certain implementations, the techniques of the invention may be applied to per-document analysis (e.g., summarization, keyword extraction, etc.) that does not require a global analysis of an entire corpus. Using lagging per-document analysis information is especially useful in scenarios where the change rate of documents is high, and the per-document analysis is expensive.

Additional Implementation Details

[0051] The described techniques for a pipelined architecture for global analysis and index building may be implemented as a method, apparatus or article of manufacture using standard programming and/or engineering techniques to produce software, firmware, hardware, or any combination thereof. The term "article of manufacture" as used herein refers to code or logic implemented in hardware logic (e.g., an integrated circuit chip, Programmable Gate Array (PGA), Application Specific Integrated Circuit (ASIC), etc.) or a computer readable medium, such as magnetic storage medium (e.g., hard disk drives, floppy disks, tape, etc.), optical storage (CD-ROMs, optical disks, etc.),

volatile and non-volatile memory devices (e.g., EEPROMs, ROMs, PROMs, RAMs, DRAMs, SRAMs, firmware, programmable logic, etc.). Code in the computer readable medium is accessed and executed by a processor. The code in which various implementations are implemented may further be accessible through a transmission media or from a file server over a network. In such cases, the article of manufacture in which the code is implemented may comprise a transmission media, such as a network transmission line, wireless transmission media, signals propagating through space, radio waves, infrared signals, etc. Thus, the "article of manufacture" may comprise the medium in which the code is embodied. Additionally, the "article of manufacture" may comprise a combination of hardware and software components in which the code is embodied, processed, and executed. Of course, those skilled in the art will recognize that many modifications may be made to this configuration without departing from the scope of the present invention, and that the article of manufacture may comprise any information bearing medium known in the art.

[0052] The logic of FIGs. 6A, 7A, and 9A describes specific operations occurring in a particular order. In alternative implementations, certain of the logic operations may be performed in a different order, modified or removed. Moreover, operations may be added to the above described logic and still conform to the described implementations. Further, operations described herein may occur sequentially or certain operations may be processed in parallel, or operations described as performed by a single process may be performed by distributed processes.

[0053] The illustrated logic of FIGs. 6A, 7A, and 9A may be implemented in software, hardware, programmable and non-programmable gate array logic or in some combination of hardware, software, or gate array logic.

[0054] FIG. 10 illustrates an architecture of a computer system that may be used in accordance with certain implementations of the invention. For example, client computer 100, server computer 120, and/or operator console 180 may implement computer architecture 1000. The computer architecture 1000 may implement a processor 1002 (e.g., a microprocessor), a memory 1004 (e.g., a volatile memory device), and storage 1010 (e.g., a non-volatile storage area, such as magnetic disk drives, optical disk drives, a

tape drive, etc.). An operating system 1005 may execute in memory 1004. The storage 1010 may comprise an internal storage device or an attached or network accessible storage. Computer programs 1006 in storage 1010 may be loaded into the memory 1004 and executed by the processor 1002 in a manner known in the art. The architecture further includes a network card 1008 to enable communication with a network. An input device 1012 is used to provide user input to the processor 1002, and may include a keyboard, mouse, pen-stylus, microphone, touch sensitive display screen, or any other activation or input mechanism known in the art. An output device 1014 is capable of rendering information from the processor 1002, or other component, such as a display monitor, printer, storage, etc. The computer architecture 1000 of the computer systems may include fewer components than illustrated, additional components not illustrated herein, or some combination of the components illustrated and additional components.

[0055] The computer architecture 1000 may comprise any computing device known in the art, such as a mainframe, server, personal computer, workstation, laptop, handheld computer, telephony device, network appliance, virtualization device, storage controller, etc. Any processor 1002 and operating system 1005 known in the art may be used.

[0056] The foregoing description of implementations of the invention has been presented for the purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise form disclosed. Many modifications and variations are possible in light of the above teaching. It is intended that the scope of the invention be limited not by this detailed description, but rather by the claims appended hereto. The above specification, examples and data provide a complete description of the manufacture and use of the composition of the invention. Since many implementations of the invention may be made without departing from the spirit and scope of the invention, the invention resides in the claims hereinafter appended.